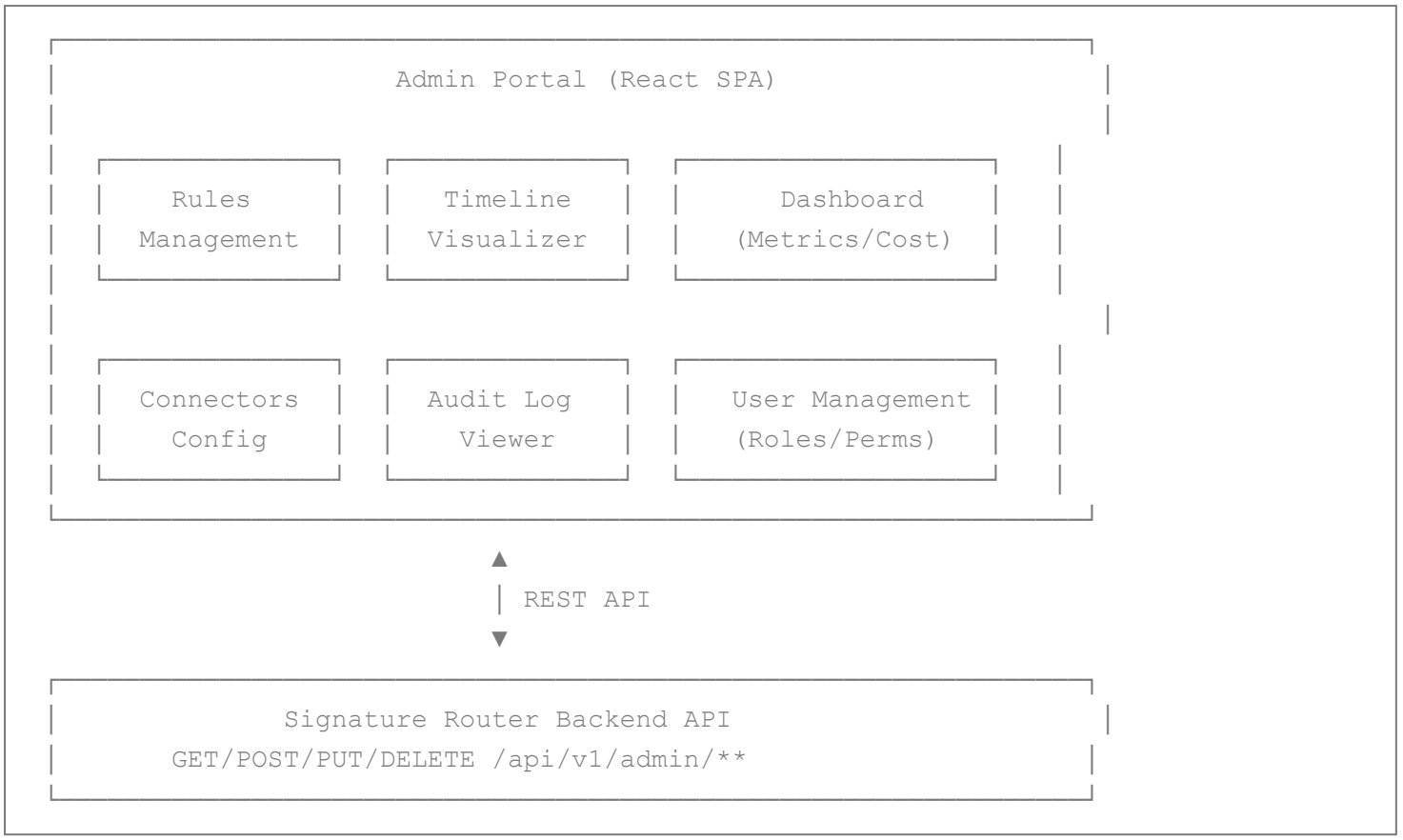
Admin Portal Architecture

Version: 1.0
Date: 2025-11-26
Status: Implementation Ready
Stack: React 18 + TypeScript + Material-UI

---

1. Portal Overview



---

2. Technology Stack

2.1 Core Framework

```
{
  "name": "signature-router-admin",
  "version": "1.0.0",
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "typescript": "^5.0.0",
    "@mui/material": "^5.14.0",
    "@mui/icons-material": "^5.14.0",
```

```
    "@mui/x-data-grid": "^6.16.0",
    "@mui/x-date-pickers": "^6.16.0",
    "react-router-dom": "^6.16.0",
    "axios": "^1.5.0",
    "react-query": "^3.39.3",
    "recharts": "^2.8.0",
    "react-hook-form": "^7.47.0",
    "yup": "^1.3.2",
    "dayjs": "^1.11.9",
    "react-toastify": "^9.1.3"
  },
  "devDependencies": {
    "@types/react": "^18.2.0",
    "@types/react-dom": "^18.2.0",
    "vite": "^4.4.9",
    "eslint": "^8.50.0",
    "prettier": "^3.0.3"
  }
}
```

2.2 Project Structure

```
admin-portal/
├── public/
│   └── index.html
├── src/
│   ├── App.tsx
│   ├── main.tsx
│   ├── components/
│   │   ├── common/
│   │   │   ├── Layout.tsx
│   │   │   ├── Navbar.tsx
│   │   │   ├── Sidebar.tsx
│   │   │   └── ProtectedRoute.tsx
│   │   ├── rules/
│   │   │   ├── RuleList.tsx
│   │   │   ├── RuleEditor.tsx
│   │   │   ├── RuleForm.tsx
│   │   │   └── SpelValidator.tsx
│   │   ├── timeline/
│   │   │   ├── RoutingTimeline.tsx
│   │   │   └── TimelineEvent.tsx
│   │   ├── dashboard/
│   │   │   ├── MetricsDashboard.tsx
│   │   │   ├── CostOptimization.tsx
│   │   │   ├── ProviderHealth.tsx
│   │   │   └── SLOMonitor.tsx
```

```
│   │   ├── connectors/
│   │   │   ├── ConnectorList.tsx
│   │   │   └── ConnectorStatus.tsx
│   │   └── audit/
│   │       └── AuditLogViewer.tsx
│   ├── api/
│   │   ├── client.ts
│   │   ├── rules.ts
│   │   ├── signatures.ts
│   │   ├── connectors.ts
│   │   └── audit.ts
│   ├── hooks/
│   │   ├── useRules.ts
│   │   ├── useSignatures.ts
│   │   └── useAuth.ts
│   ├── types/
│   │   ├── Rule.ts
│   │   ├── Signature.ts
│   │   ├── Connector.ts
│   │   └── Audit.ts
│   ├── utils/
│   │   ├── formatting.ts
│   │   └── validation.ts
│   └── theme/
│       └── theme.ts
├── package.json
├── tsconfig.json
└── vite.config.ts
```

---

3. Key Features

3.1 Rule Management

3.1.1 RuleList Component

```tsx
// src/components/rules/RuleList.tsx
import React from 'react';
import { DataGrid, GridColDef } from '@mui/x-data-grid';
import { Button, Chip, IconButton } from '@mui/material';
import { Edit, Delete, Add } from '@mui/icons-material';
import { useRules } from '../../hooks/useRules';

export const RuleList: React.FC = () => {
  const { rules, isLoading, deleteRule } = useRules();

  const columns: GridColDef[] = [
    {
```

```
    field: 'priority',
    headerName: 'Priority',
    width: 100,
    type: 'number'
  },
  {
    field: 'name',
    headerName: 'Rule Name',
    flex: 1,
    minWidth: 200
  },
  {
    field: 'condition',
    headerName: 'SpEL Condition',
    flex: 2,
    minWidth: 300,
    renderCell: (params) => (
      <code style={{ fontSize: '0.875rem' }}>
        {params.value}
      </code>
    )
  },
  {
    field: 'targetChannel',
    headerName: 'Channel',
    width: 120,
    renderCell: (params) => {
      const color = {
        'SMS': 'primary',
        'PUSH': 'success',
        'VOICE': 'warning',
        'BIOMETRIC': 'error'
      }[params.value] || 'default';

      return <Chip label={params.value} color={color} size="small" />;
    }
  },
  {
    field: 'enabled',
    headerName: 'Status',
    width: 100,
    renderCell: (params) => (
      <Chip
        label={params.value ? 'Enabled' : 'Disabled'}
        color={params.value ? 'success' : 'default'}
        size="small"
      />
```

```
      )
    },
    {
      field: 'actions',
      headerName: 'Actions',
      width: 120,
      sortable: false,
      renderCell: (params) => (
        <>
          <IconButton
            size="small"
            onClick={() => handleEdit(params.row.id)}
          >
            <Edit />
          </IconButton>
          <IconButton
            size="small"
            color="error"
            onClick={() => handleDelete(params.row.id)}
          >
            <Delete />
          </IconButton>
        </>
      )
    }
  ];

  const handleEdit = (id: string) => {
    // Navigate to edit form
  };

  const handleDelete = async (id: string) => {
    if (confirm('Delete this rule?')) {
      await deleteRule(id);
    }
  };

  return (
    <div style={{ height: 600, width: '100%' }}>
      <div style={{ marginBottom: 16 }}>
        <Button
          variant="contained"
          startIcon={<Add />}
          onClick={() => handleEdit('new')}
        >
          Create Rule
        </Button>
```

```
      </div>

      <DataGrid
        rows={rules || []}
        columns={columns}
        loading={isLoading}
        pageSize={25}
        rowsPerPageOptions={[25, 50, 100]}
        disableSelectionOnClick
        autoHeight
      />
    </div>
  );
};
```

3.1.2 RuleEditor Component

```
// src/components/rules/RuleEditor.tsx
import React from 'react';
import { useForm, Controller } from 'react-hook-form';
import { yupResolver } from '@hookform/resolvers/yup';
import * as yup from 'yup';
import {
  TextField,
  Select,
  MenuItem,
  FormControl,
  InputLabel,
  Button,
  Switch,
  FormControlLabel,
  Alert
} from '@mui/material';
import { SpelValidator } from './SpelValidator';

const schema = yup.object({
  name: yup.string().required('Name is required').min(3).max(255),
  description: yup.string().max(1000),
  priority: yup.number().required().min(0),
  condition: yup.string().required('SpEL condition is required'),
  targetChannel: yup.string().required().oneOf(['SMS', 'PUSH', 'VOICE',
'BIOMETRIC']),
  enabled: yup.boolean()
});

type RuleFormData = yup.InferType<typeof schema>;
```

```tsx
export const RuleEditor: React.FC<{ ruleId?: string }> = ({ ruleId }) => {
  const { control, handleSubmit, formState: { errors }, watch } =
useForm<RuleFormData>({
    resolver: yupResolver(schema),
    defaultValues: {
      enabled: true,
      priority: 100
    }
  });

  const condition = watch('condition');

  const onSubmit = async (data: RuleFormData) => {
    try {
      if (ruleId === 'new') {
        await createRule(data);
      } else {
        await updateRule(ruleId, data);
      }
      // Show success toast
      // Navigate back to list
    } catch (error) {
      // Show error toast
    }
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <Controller
        name="name"
        control={control}
        render={({ field }) => (
          <TextField
            {...field}
            label="Rule Name"
            fullWidth
            margin="normal"
            error={!!errors.name}
            helperText={errors.name?.message}
          />
        )}
      />

      <Controller
        name="description"
        control={control}
        render={({ field }) => (
```

```
          <TextField
            {...field}
            label="Description"
            fullWidth
            multiline
            rows={3}
            margin="normal"
          />
        )}
      />

      <Controller
        name="priority"
        control={control}
        render={({ field }) => (
          <TextField
            {...field}
            label="Priority (lower = higher priority)"
            type="number"
            fullWidth
            margin="normal"
            error={!!errors.priority}
            helperText={errors.priority?.message}
          />
        )}
      />

      <Controller
        name="condition"
        control={control}
        render={({ field }) => (
          <>
            <TextField
              {...field}
              label="SpEL Condition"
              fullWidth
              multiline
              rows={4}
              margin="normal"
              error={!!errors.condition}
              helperText={errors.condition?.message}
              placeholder="context.riskLevel == 'HIGH' && context.amount.value >
10000"
            />
            <SpelValidator expression={condition} />
          </>
        )}
```

```
      />

      <Controller
        name="targetChannel"
        control={control}
        render={({ field }) => (
          <FormControl fullWidth margin="normal">
            <InputLabel>Target Channel</InputLabel>
            <Select {...field} label="Target Channel">
              <MenuItem value="SMS">SMS</MenuItem>
              <MenuItem value="PUSH">Push Notification</MenuItem>
              <MenuItem value="VOICE">Voice Call</MenuItem>
              <MenuItem value="BIOMETRIC">Biometric</MenuItem>
            </Select>
          </FormControl>
        )}
      />

      <Controller
        name="enabled"
        control={control}
        render={({ field }) => (
          <FormControlLabel
            control={<Switch {...field} checked={field.value} />}
            label="Enabled"
          />
        )}
      />

      <div style={{ marginTop: 24 }}>
        <Button type="submit" variant="contained" color="primary">
          {ruleId === 'new' ? 'Create Rule' : 'Update Rule'}
        </Button>
        <Button
          variant="outlined"
          style={{ marginLeft: 8 }}
          onClick={() => history.back()}
        >
          Cancel
        </Button>
      </div>
    </form>
  );
};
```

3.1.3 SpEL Validator Component

```tsx
// src/components/rules/SpelValidator.tsx
import React, { useEffect, useState } from 'react';
import { Alert, CircularProgress } from '@mui/material';
import { validateSpelExpression } from '../../api/rules';

export const SpelValidator: React.FC<{ expression: string }> = ({ expression }) =>
{
  const [validation, setValidation] = useState<{
    valid: boolean;
    message?: string;
  } | null>(null);
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    if (!expression) {
      setValidation(null);
      return;
    }

    const timer = setTimeout(async () => {
      setLoading(true);
      try {
        const result = await validateSpelExpression(expression);
        setValidation(result);
      } catch (error) {
        setValidation({ valid: false, message: 'Validation failed' });
      } finally {
        setLoading(false);
      }
    }, 500);  // Debounce 500ms

    return () => clearTimeout(timer);
  }, [expression]);

  if (!expression) return null;
  if (loading) return <CircularProgress size={20} />;
  if (!validation) return null;

  return (
    <Alert severity={validation.valid ? 'success' : 'error'} sx={{ mt: 1 }}>
      {validation.valid
        ? '✓ Valid SpEL expression'
        : `✗ Invalid: ${validation.message}`
      }
    </Alert>
```

```
  );
};
```

3.2 Routing Timeline Visualizer

```tsx
// src/components/timeline/RoutingTimeline.tsx
import React from 'react';
import {
  Timeline,
  TimelineItem,
  TimelineSeparator,
  TimelineConnector,
  TimelineContent,
  TimelineDot,
  TimelineOppositeContent
} from '@mui/lab';
import { Typography, Chip, Card, CardContent } from '@mui/material';
import {
  CheckCircle,
  Error,
  Info,
  Warning
} from '@mui/icons-material';
import { RoutingEvent } from '../../types/Signature';

interface RoutingTimelineProps {
  events: RoutingEvent[];
}

export const RoutingTimeline: React.FC<RoutingTimelineProps> = ({ events }) => {
  const getEventIcon = (event: string) => {
    switch (event) {
      case 'REQUEST_CREATED':
      case 'SIGNATURE_COMPLETED':
        return <CheckCircle />;
      case 'CHALLENGE_FAILED':
      case 'SIGNATURE_EXPIRED':
        return <Error />;
      case 'FALLBACK_TRIGGERED':
        return <Warning />;
      default:
        return <Info />;
    }
  };
```

```
  const getEventColor = (event: string): 'success' | 'error' | 'warning' | 'info'
=> {
    switch (event) {
      case 'REQUEST_CREATED':
      case 'SIGNATURE_COMPLETED':
        return 'success';
      case 'CHALLENGE_FAILED':
      case 'SIGNATURE_EXPIRED':
        return 'error';
      case 'FALLBACK_TRIGGERED':
        return 'warning';
      default:
        return 'info';
    }
  };

  return (
    <Card>
      <CardContent>
        <Typography variant="h6" gutterBottom>
          Routing Timeline
        </Typography>

        <Timeline position="right">
          {events.map((event, index) => (
            <TimelineItem key={index}>
              <TimelineOppositeContent color="text.secondary">
                {new Date(event.timestamp).toLocaleTimeString()}
              </TimelineOppositeContent>

              <TimelineSeparator>
                <TimelineDot color={getEventColor(event.event)}>
                  {getEventIcon(event.event)}
                </TimelineDot>
                {index < events.length - 1 && <TimelineConnector />}
              </TimelineSeparator>

              <TimelineContent>
                <Chip
                  label={event.event.replace(/_/g, ' ')}
                  color={getEventColor(event.event)}
                  size="small"
                  sx={{ mb: 0.5 }}
                />
                <Typography variant="body2">
                  {event.details}
                </Typography>
```

```
                {event.metadata && (
                  <Typography variant="caption" color="text.secondary">
                    {JSON.stringify(event.metadata)}
                  </Typography>
                )}
              </TimelineContent>
            </TimelineItem>
          ))}
        </Timeline>
      </CardContent>
    </Card>
  );
};
```

3.3 Cost Optimization Dashboard

```
// src/components/dashboard/CostOptimization.tsx
import React from 'react';
import { Card, CardContent, Typography, Grid } from '@mui/material';
import {
  BarChart,
  Bar,
  XAxis,
  YAxis,
  Tooltip,
  Legend,
  PieChart,
  Pie,
  Cell
} from 'recharts';
import { useCostMetrics } from '../../hooks/useCostMetrics';

export const CostOptimization: React.FC = () => {
  const { data, isLoading } = useCostMetrics();

  if (isLoading || !data) return <div>Loading...</div>;

  const channelDistribution = [
    { name: 'SMS', value: data.smsCount, cost: data.smsCost },
    { name: 'Push', value: data.pushCount, cost: data.pushCost },
    { name: 'Voice', value: data.voiceCount, cost: data.voiceCost }
  ];

  const COLORS = ['#0088FE', '#00C49F', '#FFBB28'];

  const costComparison = [
```

```
    {
      name: 'Current Month',
      SMS: data.smsCost,
      Push: data.pushCost,
      Voice: data.voiceCost
    }
  ];

  return (
    <Grid container spacing={3}>
      <Grid item xs={12} md={6}>
        <Card>
          <CardContent>
            <Typography variant="h6" gutterBottom>
              Channel Distribution
            </Typography>
            <PieChart width={400} height={300}>
              <Pie
                data={channelDistribution}
                cx={200}
                cy={150}
                labelLine={false}
                label={(entry) => `${entry.name}: ${entry.value}`}
                outerRadius={80}
                fill="#8884d8"
                dataKey="value"
              >
                {channelDistribution.map((entry, index) => (
                  <Cell key={`cell-${index}`} fill={COLORS[index % COLORS.length]}
/>

                ))}
              </Pie>
              <Tooltip />
            </PieChart>
          </CardContent>
        </Card>
      </Grid>

      <Grid item xs={12} md={6}>
        <Card>
          <CardContent>
            <Typography variant="h6" gutterBottom>
              Cost by Channel
            </Typography>
            <BarChart width={400} height={300} data={costComparison}>
              <XAxis dataKey="name" />
              <YAxis />
```

```
            <Tooltip />
            <Legend />
            <Bar dataKey="SMS" fill="#0088FE" />
            <Bar dataKey="Push" fill="#00C49F" />
            <Bar dataKey="Voice" fill="#FFBB28" />
          </BarChart>
        </CardContent>
      </Card>
    </Grid>

    <Grid item xs={12}>
      <Card>
        <CardContent>
          <Typography variant="h5" gutterBottom>
            Cost Savings Summary
          </Typography>
          <Grid container spacing={2}>
            <Grid item xs={12} sm={4}>
              <Typography variant="body2" color="text.secondary">
                Total Cost This Month
              </Typography>
              <Typography variant="h4">
                ${data.totalCost.toFixed(2)}
              </Typography>
            </Grid>
            <Grid item xs={12} sm={4}>
              <Typography variant="body2" color="text.secondary">
                Savings vs All SMS
              </Typography>
              <Typography variant="h4" color="success.main">
                ${data.savingsVsAllSms.toFixed(2)}
              </Typography>
            </Grid>
            <Grid item xs={12} sm={4}>
              <Typography variant="body2" color="text.secondary">
                Avg Cost per Signature
              </Typography>
              <Typography variant="h4">
                ${data.avgCostPerSignature.toFixed(4)}
              </Typography>
            </Grid>
          </Grid>
        </CardContent>
      </Card>
    </Grid>
  </Grid>
);
```

```
};
```

3.4 Provider Health Monitor

```
// src/components/dashboard/ProviderHealth.tsx
import React from 'react';
import {
  Card,
  CardContent,
  Typography,
  LinearProgress,
  Chip,
  Box
} from '@mui/material';
import { CheckCircle, Error, Warning } from '@mui/icons-material';
import { useProviderHealth } from '../../hooks/useProviderHealth';

export const ProviderHealth: React.FC = () => {
  const { providers, isLoading } = useProviderHealth();

  if (isLoading) return <LinearProgress />;

  return (
    <Card>
      <CardContent>
        <Typography variant="h6" gutterBottom>
          Provider Health Status
        </Typography>

        {providers?.map((provider) => (
          <Box key={provider.name} sx={{ mb: 2, p: 2, border: '1px solid #e0e0e0',
borderRadius: 1 }}>
            <Box display="flex" justifyContent="space-between"
alignItems="center">
              <Typography variant="subtitle1" fontWeight="bold">
                {provider.name}
              </Typography>

              {provider.degradedMode ? (
                <Chip icon={<Error />} label="Degraded" color="error" size="small"
/>
              ) : provider.errorRate > 25 ? (
                <Chip icon={<Warning />} label="Warning" color="warning"
size="small" />
              ) : (
```

```
                <Chip icon={<CheckCircle />} label="Healthy" color="success"
size="small" />
              )}
            </Box>

            <Box sx={{ mt: 1 }}>
              <Typography variant="body2" color="text.secondary">
                Error Rate: {provider.errorRate.toFixed(2)}%
              </Typography>
              <LinearProgress
                variant="determinate"
                value={Math.min(provider.errorRate, 100)}
                color={provider.errorRate > 50 ? 'error' : provider.errorRate > 25
? 'warning' : 'success'}
                sx={{ mt: 0.5 }}
              />
            </Box>

            <Typography variant="caption" color="text.secondary">
              Last health check: {new
Date(provider.lastHealthCheck).toLocaleString()}
            </Typography>

            {provider.degradedMode && (
              <Typography variant="caption" color="error" display="block">
                Degraded since: {new
Date(provider.degradedSince).toLocaleString()}
              </Typography>
            )}
          </Box>
        ))}
      </CardContent>
    </Card>
  );
};
```

---

4. API Client

```
// src/api/client.ts
import axios, { AxiosInstance } from 'axios';

const API_BASE_URL = import.meta.env.VITE_API_BASE_URL ||
'http://localhost:8080/api/v1';

export const apiClient: AxiosInstance = axios.create({
  baseURL: API_BASE_URL,
```

```
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json'
  }
});

// Request interceptor (add auth token)
apiClient.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('auth_token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

// Response interceptor (handle errors)
apiClient.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      // Redirect to login
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);
```

```
// src/api/rules.ts
import { apiClient } from './client';
import { RoutingRule, CreateRuleRequest, UpdateRuleRequest } from '../types/Rule';

export const rulesApi = {
  list: async (): Promise<RoutingRule[]> => {
    const response = await apiClient.get('/admin/rules');
    return response.data;
  },

  get: async (id: string): Promise<RoutingRule> => {
    const response = await apiClient.get(`/admin/rules/${id}`);
    return response.data;
  },

  create: async (rule: CreateRuleRequest): Promise<RoutingRule> => {
    const response = await apiClient.post('/admin/rules', rule);
```

```
    return response.data;
  },

  update: async (id: string, rule: UpdateRuleRequest): Promise<RoutingRule> => {
    const response = await apiClient.put(`/admin/rules/${id}`, rule);
    return response.data;
  },

  delete: async (id: string): Promise<void> => {
    await apiClient.delete(`/admin/rules/${id}`);
  },

  validateSpel: async (expression: string): Promise<{ valid: boolean; message?:
string }> => {
    const response = await apiClient.post('/admin/rules/validate-spel', {
expression });
    return response.data;
  }
};
```

---

5. Authentication & Authorization

```
// src/hooks/useAuth.ts
import { create } from 'zustand';
import { jwtDecode } from 'jwt-decode';

interface AuthState {
  token: string | null;
  user: {
    username: string;
    roles: string[];
  } | null;
  isAuthenticated: boolean;
  login: (token: string) => void;
  logout: () => void;
  hasRole: (role: string) => boolean;
}

export const useAuth = create<AuthState>((set, get) => ({
  token: localStorage.getItem('auth_token'),
  user: null,
  isAuthenticated: false,

  login: (token: string) => {
    localStorage.setItem('auth_token', token);
    const decoded: any = jwtDecode(token);
```

```
    set({
      token,
      user: {
        username: decoded.sub,
        roles: decoded.roles || []
      },
      isAuthenticated: true
    });
  },

  logout: () => {
    localStorage.removeItem('auth_token');
    set({ token: null, user: null, isAuthenticated: false });
  },

  hasRole: (role: string) => {
    const user = get().user;
    return user?.roles.includes(role) || false;
  }
}));
```

---

6. Deployment

6.1 Docker Configuration

```
# Dockerfile (admin-portal)
FROM node:18-alpine AS build

WORKDIR /app
COPY package*.json ./
RUN npm ci

COPY . .
RUN npm run build

# Production stage
FROM nginx:alpine
COPY --from=build /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

```
# nginx.conf
server {
    listen 80;
```

```
    server_name _;
    root /usr/share/nginx/html;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_pass http://signature-router-backend:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    gzip on;
    gzip_types text/plain text/css application/json application/javascript
text/xml application/xml application/xml+rss text/javascript;
}
```

Status: ✅ **COMPLETE - ADMIN PORTAL ARCHITECTURE DOCUMENTED**

Next Steps:

- Implement React components
- Set up authentication flow
- Integrate with backend API
- Deploy to production (Kubernetes/Docker)
- Configure CDN for static assets